ARMY RESEARCH LABORATORY

Optimizing the Performance of a Network File System

by Daniel M. Pressel

ARL-SR-48 December 1996

The findings in this report are not to be construed as an official Department of the Army position unless so designated by other authorized documents.

Citation of manufacturer's or trade names does not constitute an official endorsement or approval of the use thereof.

Destroy this report when it is no longer need. Do not return it to the originator.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY *(Leave blank)*	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	December 1996	Final, Sep 86–Sep 92

4. TITLE AND SUBTITLE
Optimizing the Performance of a Network File System

5. FUNDING NUMBERS

6. AUTHOR(S)
Daniel M. Pressel

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
U.S. Army Research Laboratory
ATTN: AMSRL-CI-HA
Aberdeen Proving Ground, MD 21005-5067

8. PERFORMING ORGANIZATION REPORT NUMBER
ARL–SR–48

9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)

10. SPONSORING / MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES
This work was performed under the auspices of the U.S. Army Chemical Research, Development, and Engineering Center, Edgewood Area, Aberdeen Proving Ground, MD.

12a. DISTRIBUTION / AVAILABILITY STATEMENT
Approved for public release; distribution is unlimited.

12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

This report summarizes conclusions drawn from work (primarily as a systems and network administrator) conducted by the author when he was still employed at the U.S. Army's Chemical Research, Development, and Engineering Center (CRDEC). It will discuss a number of considerations which can affect the performance of computer systems running a variant of Sun Microsystem's Network File System (NFS). Additionally, the potential for enhancements to the TCP/IP standard will also be raised. This report assumes that the reader already has some familiarity with NFS and, hopefully, some knowledge of networking using TCP/IP. It is hoped that this report will help other system and network administrators use NFS, while at the same time encouraging developers of NFS to come up with more robust implementations.

14. SUBJECT TERMS	15. NUMBER OF PAGES
NFS, networks, file, computers	27
	16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED	SAR

INTENTIONALLY LEFT BLANK.

ACKNOWLEDGMENTS

The author wishes to thank his colleagues throughout the U.S. Army Research Laboratory and the Edgewood Research, Development, and Engineering Center of the Chemical Biological Defense Command for their help in the preparation of this report.

INTENTIONALLY LEFT BLANK.

TABLE OF CONTENTS

		Page
	ACKNOWLEDGMENTS	iii
1.	INTRODUCTION	1
2.	UNDERSTANDING HOW NFS AFFECTS DATA INTEGRITY	2
3.	MAINTAINING DATA SECURITY IN AN NFS ENVIRONMENT	4
3.1	Data Security and the NFS Client	5
3.2	Data Security and the Network	7
4.	BACKGROUND INFORMATION	9
5.	OPTIMIZING THE SPEED OF NFS	11
5.1	The Effect of Network Speed on the Performance of NFS	12
5.2	How the File Server Affects NFS Performance	13
5.3	Dropped Packets and How They Affect NFS Performance	14
5.4	The Source of the Dropped Packets	16
5.5	Putting It All Together	17
6.	DECIDING HOW TO USE NFS	18
7.	SUGGESTIONS FOR FUTURE DIRECTIONS	20
8.	CONCLUSIONS	21
	BIBLIOGRAPHY	23
	DISTRIBUTION LIST	25

INTENTIONALLY LEFT BLANK.

1. INTRODUCTION

Sun Microsystem's Network File System (NFS) has become a de facto standard for sharing portions of a file system between two or more multiuser computer systems (there are also implementations of NFS which allow a personal computer (PC) to share a portion of a larger computer's file system). This report will discuss some limitations of NFS and will suggest practical short-term solutions which anyone planning to use NFS should be able to use to improve performance. Since some of NFS's limitations result from the limitations of Transmission Control Protocol/Internetwork Protocol (TCP/IP), the only long-term solutions to these problems will be based on the continuing evolution of that group of standards. In that light, the author will make some initial suggestions which it is hoped will help get the ball rolling.

There are three basic aspects to the performance of any system:

(1) In most cases any acceptable system must be reliable, and in the case of a disk drive, or something which looks to the computer like a disk drive, this becomes of paramount importance.

(2) When it comes to networked computer systems, there is also the question of security. This usually refers to both the questions of the ease with which an unauthorized party can access the data and the safeguards against unauthorized tampering or destruction of data.

(3) Only after satisfying the first two criteria can one really start to talk about speed. On the other hand, if one fails to consider speed in the equation, one may be left with a system which theoretically works but fails to meet the needs of the users.

While this report is primarily concerned with the last of these three issues, it will spend some time discussing each of these points.

In turn, there are several major factors which can affect the speed of any networked client-server-based system. Some of these factors are as follows:

(1) The specifications of the network.

(2) The specifications of the network protocols on which the system is based.

(3) The characteristics of the server(s) and the client(s).

(4) The additional duties that are assigned to the server.

(5) The number of clients there are and the way they are being used.

(6) Finally, the way the client-server-based system (in this case NFS) fits into the total picture.

This report will be looking at each of these points, both individually and in combination, in an attempt to develop a better understanding of what the performance of different system and network configurations is likely to be.

2. UNDERSTANDING HOW NFS AFFECTS DATA INTEGRITY

In theory, using NFS should not in any way affect the integrity of the data. Unfortunately, this theory is closely related to the theory that the electronics never fail or get into a strange state. It also ignores the fact that there are options (frequently used options) which trade off reliability vs. speed. Therefore, now that we have dispensed with the theory, what is the reality of the situation?

Before proceeding, it is important to understand how NFS is implemented on Sun Microsystem's computers (other UNIX-based systems are likely to have similar implementations, while the implementation of NFS on non-UNIX operating systems is likely to have significant differences). Sun modified the kernel of its operating system to provide hooks that could be used to connect to the user-level software that actually implements NFS. This design maintained the UNIX philosophy of minimizing the amount of code that actually resides in the kernel so that it is easier to experiment with new features and new implementations of existing features. The user-level software which implements NFS consists of two daemon processes (jobs which run in the background and provide services to other jobs)—NFSD and BIOD. NFSD runs only on the server and responds to requests from the client, while BIOD runs on the client (although one will normally also find it running on the server) and handles requests for disk access from other processes running on the client by generating the appropriate requests to the server.

For a variety of reasons, this process tends to be significantly slower than normal disk accesses. To improve upon performance, Sun introduced the concept of the caching of data. This is not a new concept;

for many years now higher performance systems have been using disk caching. The difference here is that in addition to the disk cache which the file server might maintain, the client would maintain its own cache of NFS data. In this way, whenever possible the client would be able to satisfy read requests directly from cache. Since UNIX will normally handle all read requests synchronously and all write requests asynchronously (the process continues on doing its thing without waiting for the data to actually be written to disk), the use of cache can have a dramatic effect on performance.

Since many of today's computers incorporate caching schemes in their input/output (I/O) hierarchy and/or their memory hierarchy, this use of cache may not seem to cause any problems. The difference here is that one may potentially have several clients all mounting the same file system, and each maintaining its own cache, without any mechanism for enforcing cache coherency.* Therefore, if one client (or the server) modifies a file, the other clients may not immediately see the change. In practice, one rarely finds this to be a problem with data files, although it might be a problem with heavily accessed databases. Unfortunately, directories are a totally different story. Commonly accessed directories (e.g., /usr/local) may be accessed so often (e.g., by the shell in an attempt to find the programs one is trying to run) that they are never purged from cache. Therefore, if a new program is added to one of these directories or, alternatively, an existing program is deleted, it may take a long time for all of the clients to see the change. In extreme cases, it may even be necessary to unmount and remount the affected partition on each client (or alternatively to reboot all of the clients). Since this is a very unfriendly process, some administrators will only make changes that affect such directories at night or on weekends, with the hope that by the next morning all of the clients will see the change.

If this were the only problem, things would probably be fine. Unfortunately, there are several other considerations which can affect data integrity. The earlier versions of UNIX (mostly pre-POSIX, although some of this may apply to newer versions of UNIX if they still do not implement full POSIX compliance) did not support concepts such as file locking, record locking, automatic clearing of the SETUID bit when a file is modified, etc. More recent implementations of UNIX have added a number of these features. Unfortunately, there tend to be two problems with these additions. The first problem is that different systems may have different capabilities, so that if one brand of computer mounts a file system from

* The issue of cache coherency refers to the problem that if the same information is maintained in more than one location, it might be changed in one place but remain unchanged (at least for some finite length of time) in another place. When this happens, there is a chance that a request for information will be satisfied from outdated data. This can be a significant problem for any system involved in distributed processing, since the amount of effort put into maintaining cache coherency tends to be inversely proportional to the performance of the system.

another brand, it is not clear which (if any) set of rules will apply. Secondly, some of these modifications were made by adding tables to the kernel, so that the information is stored internally to the operating system rather than as part of the file system (this is especially a problem with the various forms of file locking). The net effect is that one cannot count on NFS to properly honor these requests.*

The final issue relating to data integrity is not unique to NFS, but NFS's distributed nature tends to magnify the problem. Computers and operating systems are not perfect. Boards can get locked up, and data structures can become corrupted. This can, and periodically does, result in a damaged file system and is the reason why most (if not all) versions of UNIX now come with a utility to fix damaged file systems (e.g., FSCK or FSDB). The problem with NFS is that it introduces the potential for additional points of failure. Therefore, the real problem is not that NFS is more prone to failure but rather that more complicated systems are inherently more prone to failure (although not necessarily any more so than putting a separate disk on each computer). What can make these problems seem worse though is that in some cases, an NFS failure will result in a corrupted file with no obvious corruption to the file system. Silent errors of this type can be especially difficult to detect (and therefore fix) and may be the best reason for exercising caution when using NFS to write to large mission-critical files.

In summing this up, NFS-mounted file systems are probably not as robust as normally mounted UNIX file systems but may offer benefits which justify what is normally a small additional risk. There are, however, circumstances in which this is not the case. Probably the most obvious area of concern is with large multiuser databases (especially those allowing concurrent updates). Once one better understands the limitations of NFS, one should be able to design a configuration which will meet most users' needs.

3. MAINTAINING DATA SECURITY IN AN NFS ENVIRONMENT

When discussing security, one should always keep two concepts firmly in mind. The first of these is that security is a relative, rather than an absolute, concept. In other words, short of putting the computer in the middle of Fort Knox, or some other hypersecure facility, and restricting it to functioning in a single user mode with no outside connections, there is no such thing as a totally secure system! The

---

* It is true that Sun has added additional capabilities to its operating system to allow programs to avoid some of these problems. Unfortunately, some of these modifications involve nonstandard system calls which are probably not used very often and may not be supported by other vendors. Therefore, while one may be able to use these capabilities in locally written software, one should not count on their use in commercially available software (unless the vendor specifically states that it used these features).

other concept is that security costs money. This refers both to the cost of building a more secure system (e.g., buying encryption devices or software) and the decrease in performance and user friendliness which normally accompanies these efforts.

In this section, we will be discussing how the following types of activities interact with the security safeguards built around UNIX and NFS:

(1) Normal users on an NFS client attempting to access someone else's data.

(2) Root on an NFS client attempting to access files owned by root.

(3) Root on an NFS client attempting to access files owned by a normal user, although the user may not have an account on that client.

(4) Someone with physical access to an NFS client (e.g., a workstation on someone's desk) but without root (or possibly any) access to that client.

(5) Someone with direct access to the network cable the client is on.

(6) Anyone with access to the network that the computers are connected to, although they may not have anything else in common with the users of those computers.

By considering these cases, one will be able to see that while, in general, NFS-mounted file systems are no less safe than putting a local disk on someone's UNIX workstation or PC, it may not be any safer. One will also note that, in general, NFS-mounted file systems are less secure than nonnetworked file systems maintained on the traditional computer center computer.

3.1 Data Security and the NFS Client. From the standpoint of a normal user, NFS attempts to maintain the normal UNIX safeguards against unauthorized access and tampering. This is not to say that such a user cannot misbehave, rather that for most practical purposes NFS will not make it any easier to misbehave. One should keep in mind that since a normal user of an NFS client is more likely (compared to users of large centralized computer systems) to fall into one of the other categories that will be discussed, one may need to exercise greater caution when granting access to the NFS clients.

From the start, the designers of NFS realized that there was a fundamental problem with privileged access on a client. Since NFS considers a user on a client to be equivalent to the user on a server (based on userid numbers), and all UNIX-based systems have a superuser called root, by implication anyone with root access to a client would have unlimited access to the NFS-mounted partitions. From a security standpoint, this was a big problem. Initially, Sun solved the problem by allowing the system administrator of the NFS server to turn on or off the granting of root access to all of the clients by a single command (NETROOT). As long as the number of clients was small, this worked reasonably well. One would normally leave root access turned off, until it was absolutely needed, and then only turn it on for a short period of time. Unfortunately, during that short period of time, all of the clients had root access, and there was the potential for someone to write a sleeper program which would wait around for root access to be enabled, and then to march in and do its thing. More recent versions of NFS allow one to specify specific systems to be granted root access (under SunOS 4.0.3 and SunOS 4.1.1, this is done using the /etc/exports file). Unfortunately, on at least some systems it appears to be necessary to reboot the file server if one wishes to change the list of computers with root access. Therefore, there is the tendency to leave this access turned on for long periods of time (if not permanently), which may open up opportunities for other people to break in.

Assuming that root on a NFS client is turned off, all attempts by root to access any file on an NFS-mounted partition will be translated into a request from the dummy user NOBODY. Therefore, one might assume that root would be unable to access any files for which permission to OTHER has been denied. Before coming to that conclusion, one should remember that Root can SU (substitute user) to any other valid user on the system without knowing that person's password. Therefore, one should conclude that root can, in fact, access the files of any valid user on the NFS client(s) for which that person has root access. What about the files of other users? Remembering that root is free to add new users to the password file on any system that person has root access, one can conclude that an untrustworthy system administrator could obtain virtually unlimited access to an NFS-mounted partition (they still do not have unlimited access to the files owned by root). One might counter that this is not a big problem, since all system administrators are assumed to be trustworthy. Unfortunately, many installations have different standards for the system administrators of individual workstations than they have for the system administrators who work at computer centers (which is where the file servers will normally reside). Additionally, as soon as one places work stations where people can get their hands on them, there is the potential for someone to break in.

As was just stated, it can be significantly easier to gain root access to a workstation than to a computer center system. One might be surprised to hear this, since they both run multiuser operating systems (possibly even the exact same version of the operating system), and this holds true even if they are administered with the same care. Without physical access to a computer, one needs to either break through, or find a way to go around, the password system in order to obtain root access to the computer. When one has physical access to a computer, the story can be quite different. With physical access, it is usually much easier to take a system down and bring it back up in single-user mode. On many workstations, one does not even need to know any passwords to do this (even on those workstations where a password is required, it can be fairly easy to bypass that requirement). The main thing that keeps one from doing this on a computer center computer is that it tends to inconvenience a lot of people and, therefore, is rapidly discovered. Since workstations are frequently single-user systems, if no one is logged in at the console, it might never be detected, and since the break in would normally be performed from the console, it is easy to ensure that this condition is met. Once the computer has been booted in single-user mode, one has unlimited access to the local computer and can modify the password file in any of a number of ways. Then the system can be brought up in multiuser mode with the normal mounting of partitions (including those mounted via NFS). Now, should the hacker desire, he/she can gain full root access to the machine and in the process gain substantial access to the NFS-mounted file systems using the techniques that have already been discussed. This demonstrates that it is extremely important to control the physical access to work stations running NFS if one is to avoid potentially serious breaches of security, although it can also be argued that if the computers are located in a secure building with relatively nonsensitive material on the computers, this may not be a big problem.

3.2 Data Security and the Network. Now more surreptitious modes of access will be discussed. If the network cable is easily tapped, as is frequently the case with both thick and thin Ethernet, then it is possible to place an unauthorized system on the network. With an unauthorized system, it is frequently easier to run jobs which access the network in unusual ways (on authorized systems, there is a chance that the system administrator might find out what is being done). One such esoteric job is to listen to all of the traffic which comes over the network and record those portions of the traffic which are interesting. Of course, this type of attack is not limited to NFS traffic, but it might prove to be even more useful than intercepting unencrypted passwords since one does not have to physically log into the system under attack (unauthorized logins can be easy to spot on a properly run system). Taking this one step further, a knowledgeable hacker might even be able to generate spurious requests which appear to be coming from another system on the network.

There are several ways to defeat these types of attacks. Some people take great precautions to protect the cables, while others run sophisticated software aimed at detecting the network interruptions which are likely to occur when someone taps into the network for the first time. Unfortunately, both of these techniques can cost money and consume valuable manpower. Another technique which may be easier to use, involves data encryption. Here one has three ways to approach the problem. One can encrypt the entire network, one can use the security features built into some versions of NFS, or one can leave it up to the individual users to decide which files are important enough to encrypt using the software that comes with the system. Each of these solutions has its pros and cons and should be viewed as a tradeoff between the cost of protecting the system and the cost of not protecting the system. For those systems which are located in secure buildings and only process relatively nonsensitive material, the last approach is likely to be the most efficient solution to the problem.

Finally, this leaves the case of attempts to access the data over the network from remote sites. Here, there are two cases about which to worry. The person out for a joy ride, so to speak, will probably only go for systems which can easily be attacked. Therefore, any of a number of relatively simple precautions may be adequate to stop this type of break-in. On the other hand, it is doubtful that any system will be able to stand up to a dedicated hacker for long (although there are some people who feel that using encrypted networks may be a workable, albeit expensive, solution to this type of attack). For this reason, only the first case will be considered.

From the author's experience, it is clear that there are system/network administrators out there who do not fully understand how NFS's security precautions are implemented (only the most basic aspects of these precautions will be discussed here, since in a multivendor environment those may be the only ones on which one can count). By now, most system/network administrators have a basic idea of how rlogin and rcp work (from a security standpoint), with their reliance on files such as /etc/hosts.equiv, /.rhosts, and $HOME/.rhosts. There are some individuals in this line of work who assumed that NFS would rely on these same files or at least that one could not do an NFS mount from a system which does not appear in the server's /etc/hosts file. Neither of these assumptions is correct. NFS security is based on two parts. The first part is permission to mount a partition, while the second part is the standard UNIX security precautions for access to individual files. Since it has already been shown that once an NFS mount has been established, an unscrupulous system administrator may be able to gain improper access to other people's files, it is clear that NFS security depends on the ability to prevent unauthorized mounting of partitions.

Probably the easiest way to prevent unauthorized mounts, and one that is actually recommended in some security bulletins, is to remove the file /etc/exports from the server. Unfortunately, this invalidates all attempts to perform an NFS mount off of that server. In other words, the server is no longer a server. Since this report assumes that there is a good reason for having an NFS server, it will also assume that this is not an acceptable solution.

If this file must exist, then what should be in it? In its most basic form, the file /etc/exports contains a list of partition names (actually the name of the directory on the server, which was used as a mounting point, when the partition was mounted), one per line. This states that any system anywhere on the network can mount these partitions, even if the system does not appear in the file /etc/hosts. If one is not connected to the Internet (or some other large network), this may be sufficient, but for most systems this is a massive security hole. The next simplest form of this file is to follow each partition name with a list of systems allowed to access that partition. This is a significant improvement but has some serious limitations to it. For example, this means that all of the clients have the same level of permissions. Also, it means that if the server does not support the NETROOT command, or something equivalent to it, there is no way to grant root access should that become necessary.

Starting with the 4.X.X versions of SunOS, Sun enhanced the format of the /etc/exports file to allow a much higher level of specificity in the granting of access to a server's partitions. Unfortunately, not all systems support these enhancements. Additionally, the new format can get rather complicated to use if one is dealing with a large number of clients. Depending on one's needs, the additional complications associated with this new format may not be justified. In any case, it is important to realize that this file is the first line of defense, and some care should be taken in setting it up and maintaining it.

4. BACKGROUND INFORMATION

Prior to discussing how to optimize the speed of an implementation of NFS, it may be helpful to more fully understand the networking technology behind it. While it is theoretically possible to implement NFS (or something that serves the same function) using any of several common standards (e.g., TCP/IP, OSI interconnect, or the Xerox Network Systems), the only implementations on which the author has seen it run are based on the TCP/IP standard. Therefore, from now on, only this networking standard will be discussed.

TCP/IP consists of a series of user-level software, most of which is layered on top of two different protocols—Transmission Control Protocol (TCP) and User Datagram Protocol (UDP) (these will be discussed later)—which in turn are layered on top of the Internetwork Protocol (IP). Finally, the IP is layered on top of the physical hardware and the associated drivers (e.g., Ethernet). The IP layer specifies what is known as an unreliable connectionless datagram. Its purpose is to contain routing information which allows a packet to travel from one machine to the next and for the packet to contain a data section which will normally consist of either a TCP or UDP packet. There is no guarantee that an IP packet will ever make it, so if the packet is important, software at one of the higher layers must keep track of which packets have been received and which ones require retransmission. Additionally, there is the chance that duplicate copies of a packet will arrive. Again, it is the responsibility of the higher layers of software to deal with this issue.

TCP is normally referred to as a reliable stream protocol. It is reliable in the sense that this layer accepts responsibility for keeping track of which packets have arrived (throwing away duplicates and requesting/initiating the retransmission of missing packets) as well as some associated issues relating to flow control. Stream refers to the treatment of the data as a continuous byte stream without any boundaries between messages. Some common applications of this protocol are the RCP, RLOGIN, TELNET, and FTP utilities commonly found on most UNIX work stations today. The main disadvantage of TCP is that it does so much hand holding that in an environment with reliable transmission of data, it can be considered to be overkill. Unfortunately, this overkill has a cost associated with it, so some software developers wanted a simpler protocol with less system overhead.

UDP is very similar in function to the IP on top of which it is layered. With this protocol, it is up to the user-level software to deal with issues of reliability. Additionally, the basis for this protocol is fundamentally different from that of TCP. While TCP is based on the concept of a byte stream, UDP is based on the concept of a message, where each message will reside in a single UDP packet. Additionally, while each TCP packet fully identifies both the sender and the recipient of each packet, UDP packets need only identify the recipient. As such, they are perfect for transmitting small amounts of information (e.g., time stamps) or requests for service from a large number of clients, providing no one client makes a large number of requests (e.g., the Domain Name system). Its simplicity also makes it perfect for transferring information to systems of limited capabilities (possibly because the systems are special-purpose systems and possibly because the system is in the process of bootstrapping itself). Common examples of this latter use are Network Disk (ND), Trivial File Transport Protocol (TFTP), and NFS.

The use of UDP packets to transfer large quantities of data on a recurring basis represents an interesting design decision. Since UDP packets are by definition unreliable, the additional code necessary to produce a reliable transfer protocol standard must be included in the user-level code (in the case of NFS, this refers to the BIOD and NFSD processes previously discussed). In addition to the complexity this adds to the user-level code, there is a more subtle problem this can cause. If packets are dropped or duplicated, it takes significantly more central processing unit (CPU) time to handle these problems with user-level code than inside of the kernel of the operating system where most UNIX-based implementations of TCP deal with these problems (the extra CPU time is the result of both additional context switching and additional switching between the kernel and user-level code). On the other hand, if very few packets are either being dropped or duplicated, then the lower overhead of the UDP packets may actually save time.

Now on to the question of flow control. For a variety of reasons, a talker may need to be told to slow down. In the case of TCP packets, this can happen in one of two ways. If the slow point is the recipient, the recipient controls a window, indicating what data it is willing to accept. The sender is not supposed to send any packets containing data which lie outside of that window. By controlling the size of this window, and only adjusting its location when the recipient is capable of receiving more data, the recipient can control the rate at which packets enter the network. It is also possible for some other part of the network to become saturated (e.g., a gateway, where the primary reference is to the use of a general purpose computer as a router), and this can result in an ICMP Source Quench message. This message is normally intercepted by the networking subsystem of the kernel. Unfortunately, this message is normally only sent in response to TCP packets, and normally only the TCP part of the networking subsystem is affected by its receipt. Presumably, because this is considered to be an issue of reliability, it is considered to be a user-level issue for UDP packets, but since generators of UDP packets do not establish permanent connections, it is not clear which process(es) should be notified of the receipt of the ICMP source quench. As a result, the UDP packets continue to be generated full force, but the choke point is free to drop (throw away) any packets with which it cannot deal. This lack of flow control can result in a higher than expected number of dropped packets; this will be discussed further in following text.

5. OPTIMIZING THE SPEED OF NFS

When trying to optimize the speed of NFS, the three main considerations are the maximum speed of the network, the ability of the server to keep up with demand, and the minimization of the number of

dropped packets. The following subsections address these considerations and the way they affect the overall speed of the system.

5.1 The Effect of Network Speed on the Performance of NFS. Clearly, NFS cannot transmit data any faster than the maximum data rate of the network (strictly speaking, one might be able to do somewhat better if NFS used data compression algorithms, but at the present it does not use them). The first problem is to determine what that data rate is. If one is talking about Ethernet, some sources indicate that one should not rely upon more than one third of the theoretical bandwidth of the Ethernet standard (otherwise the probability of collisions, with the resultant loss in performance, becomes unacceptably high). If the network has a gateway in it, can the gateway operate at the maximum speed of the network (frequently the answer is no)? On a network composed of two or more segments, is the talker on the slowest segment (as measured by the peak data transfer rate and assuming that some of the segments are slower than others)? If not, are there sustained periods of high transmission rates sufficient to overwhelm the gateway's ability to act as a buffer?

In addition to the questions of maximum data rate as it applies to a single talker, there is also the question of maximum available data rate, when one takes into account the probability of multiple talkers (some of whom will have nothing to do with NFS). One must then compare this data rate to the job at hand. If one is using NFS to infrequently access relatively small files (much less than 1 MB (megabyte) in size), an effective maximum data rate of 56 kb/s (kilobits/second) (or possibly even less than that) might be acceptable. On the other hand, if one is using NFS to provide access to software, and the software is 15 MB in size (that is not the size of the core image but the size of the actual executable), it might take 1–2 min to start that software up when running NFS over Ethernet. Compare this to nearly 1 hr (possibly more) at 56 kb/s, and one can see the importance of making sure that the network is up to the job at hand.

Similarly, a data rate which might be acceptable when dealing with files which are only accessed a few times a day might not be adequate for scratch files or other environments which rely on heavily accessed files. Also, remember that since access to NFS-mounted files is nearly always slower than accessing a local disk, one should only use NFS to mount partitions with low to moderate levels of activity. Note, the activity level in question is based on the partition, not the file. It does not matter if one is reading 10,000 B from 1,000 files or 1,000,000 B from 10 files every minute. Either way, this is still a very heavy level of activity.

5.2 How the File Server Affects NFS Performance. Clearly, if the file server has a slow disk subsystem, then its performance as a file server will be anything but startling. One might go on to ask, are there any other parts of the file server which will significantly affect the performance of NFS? Once again, the answer is yes. Tests performed on Sun 3 work stations (including file servers) have shown that they were incapable of either receiving data from an Ethernet-based network or transmitting data to that network using FTP at anything approaching the saturation level for Ethernet (normally assumed to be about 300 kB/s, although higher rates can be achieved under optimal conditions, this also assumes that one is using software supplied by Sun).* One might object to this observation, since FTP is based on TCP packets, while NFS is based on UDP packets, but the author has observed the same behavior with UDP-based software (e.g., SPRAY). Further studies have shown that the limitation with these computers was with the processor's ability to handle that much data, not with the network connection. Therefore, it is clear that one needs to select a file server with enough CPU power to do its job.

After having selected a fast file server, one can look at the effect that the choice of network connection has on NFS performance. Some work stations, which in theory should make a very nice NFS server for a small installation, share hardware between the SCSI controller and the network interface. The result of this is that while in theory they are capable of commendable disk performance and network performance, they cannot do both at the same time. Therefore, unless one can use a separate disk controller (possibly a second SCSI controller) or a separate network connection (on Sun workstations this might mean using an ie1 card instead of the le0, ie0, or ec0 with which the workstation normally comes, such a machine will probably not make a good file server. Another problem to note is that some larger machines (e.g., Sun 490) support multiple Ethernet ports, but that does not mean that they all perform equally well. Generally, the ports directly connected to the CPU board will be the fastest (presumably because they do not need to access the system bus), while older boards seem to perform the worst (presumably they either have slower chips or smaller on board buffers). From the standpoint of a file server, the speed of the Ethernet port directly translates into the number of incoming packets which are dropped.

Finally, it should be noted that for truly demanding applications there are other aspects which may be considered (e.g., the specifications for the system bus(es), the implementation of the networking

* This is based on work performed by Jacobsen as referenced by Stevens (1990). (Stevens, W. R. UNIX Network Programming. Englewood Cliffs, NJ: Prentice-Hall, 1990.)

subsystem in the operating system, etc.). From this discussion, it should be clear that the choice of file server and even file server configuration can have a significant impact on performance. The next section will show why this might be even more important than one might think at first glance.

5.3 Dropped Packets and How They Affect NFS Performance. There are essentially three ways in which a data packet can be dropped once it leaves the talker.

(1) It may become corrupted (e.g., from electrical interference).

(2) The sender may be talking faster than the recipient, or an intermediate gateway, can listen.

(3) If one is using Ethernet, there may be a collision requiring retransmission (regardless of the packet type, this should be detected and handled by the kernel).

Since NFS is based on UDP packets, it is the responsibility of the NFS software to deal with the first two possibilities. It should be clear that regardless of the software being used, if packets are being dropped, the performance of the software will be affected. To understand the extent of the degradation, it is important to understand how the software deals with this problem.

NFS attaches a sequence number to each packet. The sequence number allows the recipient to detect duplicate packets, and to generate a positive acknowledgement of the receipt of a packet. It also allows it to rearrange the packets should they be received out of order. NFS does not attach its own checksum to each packet, nor do most implementations make use of the UDP checksum, since the reliability of the network is usually sufficient to eliminate the problem of corrupted packets. Once a packet has been received, the recipient generates an acknowledgment, which will hopefully reach the sender. Meanwhile, each time the sending NFS system generates a packet, it starts a timer. If for any reason, an acknowledgment has not been received before the timer expires, a new packet will be generated.

So far, this is not significantly different from what the TCP subsystem in the kernel does, except that it is being done by user-level software. At this point though, there are several important differences between the functioning of NFS- and TCP-based software. The first such difference is what happens while the sender is waiting for an acknowledgment. TCP uses the concept of a sliding window. Therefore, only a limited number of packets may be in transit at any one time, and the recipient controls

what that number is. In contrast to this, there are no such limits imposed on UDP-based software, and apparently NFS does not support this concept on its own. From this, one can conclude that while in theory NFS should be able to maintain higher data transmission rates (this is based on the assumption that TCP-based software might have to periodically stop and wait for the acknowledgments to return), should the network become overloaded, NFS might make matters worse.

The second difference between the way NFS- and TCP-based software functions is the question of how long to wait for the acknowledgments. Initially, NFS always waited a specific number of milliseconds, which was embedded in the code. Unfortunately, this caused a great deal of trouble when using NFS over long distances, since in many cases it was physically impossible to receive a response in the time allotted. More recent versions of NFS maintain the concept of a fixed amount of time for timeouts but now allow the system/network administrator to specify what that value should be. This works better but can have the following problems associated with it:

(1) Not all system/network administrators know about this option or how to properly use it, so most NFS installations are still using the default value.

(2) It can be difficult to determine an appropriate timeout value to use with distant systems, especially if there are intervening gateways. This problem is made more challenging in that determining the appropriate value might depend on the total level of network activity.

(3) Excessively long timeouts can negatively impact performance, since it takes a lot longer to declare a packet lost. This can be very important if some of the clients are close to the server, while others are hundreds (or even thousands) of miles away.

In contrast, newer implementations of TCP tend to use adaptive timeouts. This means that if the sender starts to fail to receive acknowledgements within the specified period of time, it will slowly extend the timeout period. Since the concept of a sliding window limits the number of packets that can be in transit at any one time, there is no real danger that this could result in the network being flooded with packets.

Once again it should be pointed out that processes using UDP packets are normally not informed that the network is overloaded and that there is a need to decrease the rate of transmission. While it should

be possible for NFS to infer that it should decrease its transmission rate from the rate at which packets are being dropped, the author has seen no evidence that this is what happens. The result of this is that if the NFS client suddenly starts writing large amounts of data to the file server, the file server is likely to drop packets. These packets will timeout and be retransmitted along with new packets. Some, possibly many, of these packets will then timeout and need to be retransmitted. The net result of all of this can be a network storm with as many as 95+% of all packets being dropped. It is the author's experience that this can result in more than an order of magnitude decrease in throughput compared to a well-tuned system.

5.4 The Source of the Dropped Packets. Up until now, this report has simply assumed that it is possible to get a sudden rush of NFS activity without explaining how this might happen or how common it might be. To answer these question (in the context of the UNIX environment), one must first understand how UNIX systems have traditionally performed disk I/O. For this discussion, any buffering which is internal to the user's program (e.g., on many systems the standard I/O packages used with C programs will default to 8-kB buffer sizes when talking with a disk drive or other blocked media) will be ignored. Instead, only the low-level read and write commands that require immediate attention by the kernel will be considered.

When a program executes a read request, the default action is for the program to go to sleep until that request has been satisfied. This practice is known as synchronous I/O and is based on the assumption that the program cannot perform any useful work until this request has been satisfied. While it is theoretically possible to write a program which performs asynchronous reads or returns immediately if no information is pending, this is rarely done in the context of disk I/O (although it is somewhat more common in the context of terminal I/O). Therefore, one can conclude that rarely if ever will a process have multiple reads to NFS-mounted partitions pending at the same time.

Write requests are somewhat more complicated. In this case, for most software (databases are an exception) there is no fundamental reason why the write requests should be handled synchronously. Therefore, early versions of UNIX handled all write requests asynchronously, so that the operating systems could schedule the actual disk I/O in a prioritized manner aimed at minimizing both the length of time required to do a read and the amount of time spent seeking the disk (moving the read/write head from one track to another track). While some versions of UNIX now support synchronous writes, the use of synchronous writes is relatively rare.

One might think that all of this is fine and dandy, but how does it result in dropped packets? The answer is that some software can write 1 MB or more of information to disk in a very short amount of time (e.g., when copying information from a scratch document back to the permanent file). Since the writes will be done asynchronously, it is possible that the client will talk faster than the server can listen. If there is only a slight mismatch in speed (i.e., less than 50%, and preferably less than 10%), then the small number of dropped packets will only result in a small number of retransmissions. When the mismatch grows to a factor of two, it is not unusual for as many as three quarters of all of the packets to be dropped. By the time the mismatch reaches a factor of seven, 95+% of all packets may be dropped (depending on how many packets are being transmitted). When there are a large number of packets being dropped and retransmitted, it is easy to see how the system will become bogged down with a cascade of network storms.

5.5 Putting It All Together. What conclusions can be drawn from this discussion? It is important that the NFS file server not only be fast enough to honor the requests at the expected average rate of arrival, but it must be fast enough to handle any reasonable peak level of requests. Only in this way can one minimize the number of dropped packets and, therefore, avoid network storms. One aspect of this process which is not well documented is the number of NFSD processes which are running on the server. Collectively, these processes are responsible for processing the requests, and the number of these processes partially determines the maximum number of requests that can be efficiently handled at the same time. Traditionally, Sun shipped the operating system for its Sun 3 systems so that four of these processes would be started at boot time. Other platforms both from Sun and from other vendors may use a different number of NFSD processes (usually larger). If one is experiencing problems, it might be desirable to change this number (usually found in the /etc/rc.local file) and reboot the server.

If the server is running a large number of NFSD processes, or has a relatively low speed CPU, it might be necessary to restrict the access to the server. In particular, some third party vendors might suggest using the server as both an NFS server and as a compute server, or a plot server, etc. Before committing to such a proposal, one needs to carefully examine how many resources are required for each of the duties being assigned to the server. When carrying out this analysis, remember that at least as far as NFS is concerned, it is the peak load, not the average load, which matters.

Similarly, one should carefully consider the appropriateness of using an NFS file server as a gateway. Traditionally, most general purpose computer systems lacked sufficiently efficient or robust networking

subsystems to properly function as a gateway in a demanding environment. This is not to say that they would not function just fine under less demanding circumstances but rather to point out that there can be a significant amount of overhead associated with the functioning of a gateway. Therefore, if there is a significant amount of nonlocal network traffic flowing through the gateway, it may interfere with the ability of the file server to handle peak loads of NFS requests. Under those conditions, it might be better to use a different machine for the functions of a gateway.

There are a number of other situations which can also cause an NFS file server to behave poorly. In general, what one has to be concerned with is that the more demanding the level of NFS activity is (relative to the capabilities of the file server), the more likely it is that one should try to eliminate all other work from the file server. An alternative to this approach is to decrease the level of NFS activity. The next section will discuss ways of doing this in greater detail.

6. DECIDING HOW TO USE NFS

There are a number of ways in which one can use a data storage and retrieval system, be it disk, tape, or NFS. The real trick is to match the technology to the job at hand, keeping in mind that both the technology and the job are likely to be moving targets. Some organizations will use diskless clients, with everything including the operating system and the client's swap area residing on the NFS file server. Other people prefer to use dataless clients, where the client has its own disk drive with the operating system and swap partitions residing on the local drive but everything else going over the network. Many installations use fully equipped workstations but choose to use NFS file servers to provide access to infrequently used software and/or archival storage of data (in the later case, the user would have online access to the data files for one or more locations without having to tie up the local disk drives). Each of these options has both its strong and weak points, so the remainder of this section will briefly discuss some of these points. It is unlikely that any such discussion could be all inclusive, and it will be clear that this discussion makes no such pretensions. Instead, it is hoped that it will help the reader start to think about issues which are easily overlooked, so that the reader can do additional research as is appropriate for his/her own particular situation.

Assuming that one has the spare space on a file server, diskless clients might look very inviting. They have the advantages of being less expensive and easy to administer from a central site. Additionally, if one has several clients with the same architecture, one can normally share many of the same partitions

between all of the clients. Unfortunately, this can raise issues of reliability (the file server is a single point of failure, which will affect all of the clients), speed (using NFS as a swap device can be particularly inefficient), and load factor on the server. Even so, in certain environments (e.g., classroom situations), this may be a very reasonable solution. While there are some who will strongly disagree with this view, it is the author's opinion that diskless clients should not be used in a production environment (e.g., a drafting department).

Many of the objections to diskless clients can be overcome by using dataless clients. While these systems do need some local disk capacity and are, therefore, somewhat more expensive to buy, they do not necessarily need a large amount of local disk capacity (especially if the applications software is accessed over the network). This configuration is still dependent on the file server for access to each user's working files, which means that if the server goes down, all work stops. On the other hand, it also means that if a single work station goes down, the user can in theory move to another work station. Compared to diskless clients, it has several advantages, including significantly less network activity and higher throughput since swap and scratch activity will be off of local disk drives (as previously noted, local disk access can be up to 10 times faster than accessing files using NFS, and in the case of swap and scratch activity, this difference is easy to detect). On the other hand, since all of the working files are centrally located, this configuration simplifies the tasks of doing backups and physically protecting the data.

There is one point of caution with this configuration, while one would expect the scratch activity to go to the local disk drive, this is not always the case. While many programs write to scratch files located in /tmp, /usr/tmp, /usr/spool, or other similar locations, there are a number of commonly used third-party programs which place their scratch files in the user's current working directory. Since in this configuration, this will normally reside on the file server, these programs may not run any faster. Fortunately, most versions of UNIX now support symbolic links, and it is sometimes possible to use these links to force the scratch files to actually reside in /usr/tmp (or wherever seems to be most appropriate). While this might require a higher level of sophistication than some users are used to, it is the author's experience that for those users who regularly place heavy demands on a system, this level of sophistication is not an unreasonable expectation.

Fully functional diskful workstations will of course eliminate the need to worry about where the scratch files reside. On the other hand, they can require a great deal of additional hardware and effort to

set up. Additionally, each machine must be backed up individually (although this can be done over the network, there are reasons why one may not want to do that), and should the system go down, one would temporarily lose access to the data stored on that system. One way to lessen some of these problems is to use the NFS file server to store some of the application software and/or for archival data storage. Since some software may require 100 MB or more of storage (just for the software itself), if this software is not heavily used by any single system, it may be appropriate to live with a modest decrease in performance by storing it on the server. In this case, one must make sure that the available network bandwidth is adequate to handle the additional load.

The case of archival storage is an interesting area where NFS excels. Traditionally, if one lacked sufficient local disk space to store all of one's data files (and sooner or later almost all systems reach that point), one was left with the three choices of either deleting files, using removable storage media (e.g., tapes or disk packs), or postponing the day of reckoning by buying more disk drives. NFS offers another alternative. One can configure a server with cheaper drives (e.g., optical jukeboxes), which would be too slow for use as working storage. However, so long as the total rate of access to this server remains low, the temporary degradation in performance may be considered acceptable. This can be especially true if one compares it with the costs (both of hardware time and lost productivity) of using off-line storage. There is the additional advantage with this type of application in that it does not appear to have the behavior of placing all of one's eggs in the same basket. In particular, should the server go down for a few hours, few if any users might be affected (assuming copies of most active files are maintained in the user's working storage, which should be on another system). The biggest single problem with this type of strategy is the question of when and how to move files to the server. Unfortunately, this paper can give no guidance on that point, since it is best dealt with as a site-specific issue.

7. SUGGESTIONS FOR FUTURE DIRECTIONS

It is clear that in the future there will be an ever increasing need to reduce the potential for NFS-induced network storms. There are two potential solutions to this problem. The first is the VMTP initiative, which has been undergoing development and testing for some time but as far as the author knows is not yet in common use. This project is based on the concept of developing a reliable datagram, which would have a lighter overhead than TCP packets and would incorporate more efficient algorithms for dealing with dropped packets than do current implementations of NFS (as well as other commonly used UDP based software).

The other alternative, which might be simpler, is to allow a system receiving a large number of UDP packets (be it a gateway, or the final recipient) to send out a Source Quench type of signal, which would be respected by the kernel of the operating system of systems generating UDP packets. This might either be addressed to a single system or would probably be even more useful if it is generated as a broadcast packet (addressed to all systems on the current segment of the network). Upon receipt of such a signal, the kernel could reduce the maximum rate at which it introduces UDP-based packets onto the network.

This leaves the problem of what does the kernel do if it receives packets from user-level software (BIOD and NFSD in this case) faster than it can introduce it onto the network. Initially, it would simply allow the requests to fill up the kernel's buffers, just as it does with pending requests to write to a local disk. If the buffer space drops below a certain level, the kernel is then left with two choices. It may either signal the user-level software to wait (possibly by putting the software into a wait state or possibly by taking some action to indicate a failure on the next attempt to write to the network), or alternatively the kernel could simply drop the packet itself. This later alternative might not seem to be very useful, but it would generate the same timeout condition the software currently is testing for (therefore, it would be totally compatible with current software) without producing any network storms. There may be still other solutions to this problem (e.g., using better algorithms in BIOD and NFSD), but what is clear is the need to advance the way things are being handled.

8. CONCLUSIONS

Clearly, with only a few exceptions NFS meets any reasonable standard for data integrity, and so long as one is careful to steer clear of those exceptions (e.g., trying to run database programs using NFS), there shouldn't be any major problems. The issue of data security is a troubling area, which affects almost every aspect of computer science these days. The author believes that if the system/network administrators execute an appropriate level of diligence, it should be possible to produce an acceptably secure environment. That leaves the challenge of keeping the system running fast and represents an opportunity for careful system analysis to prove its worth. It is the author's hope that in the future more robust implementations of NFS will make it more of a turnkey system. In the meantime, it is hoped that the concepts developed in this paper will help other system/network administrators to successfully navigate this maze.

INTENTIONALLY LEFT BLANK.

BIBLIOGRAPHY

Comer, D. E. Internetworking With TCP/IP Principles, Protocols, and Architecture. Englewood Cliffs, NJ: Prentice-Hall, 1988.

Leffler, S. J., M. K. McKusick, M. J. Karels, and J. S. Quarterman. The Design and Implementation of the 4.3BSD UNIX Operating System. Reading, MA: Addison-Wesley Publishing Company, 1989.

Nemeth, E., G. Snyder, and S. Seebass. UNIX System Administration Handbook. Englewood Cliffs, NJ: Prentice-Hall, 1989.

Santifaller, M. TCP/IP and NFS Internetworking in a UNIX Environment. Reading, MA, Addison-Wesley Publishing Company, 1991.

Stallings, W. Handbook of Computer-Communications Standards. Vols. 2 and 3, New York: MacMillian Publishing Company, 1987.

Stevens, W. R. Advanced Programming in the UNIX Environment. Reading, MA: Addison-Wesley Publishing Company, Inc., 1992.

Sun Microsystems. System and Network Administration. Part Number: 800-3805-10, March 1990.

Sun Microsystems. Network Programming Guide. Part Number: 800-3850-10, March 1990.

INTENTIONALLY LEFT BLANK.

NO. OF COPIES	ORGANIZATION
2	DEFENSE TECHNICAL INFO CTR ATTN DTIC DDA 8725 JOHN J KINGMAN RD STE 0944 FT BELVOIR VA 22060-6218
1	HQDA DAMO FDQ ATTN DENNIS SCHMIDT 400 ARMY PENTAGON WASHINGTON DC 20310-0460
1	US MILITARY ACADEMY MATH SCI CTR OF EXCELLENCE DEPT OF MATHEMATICAL SCI ATTN MDN A MAJ DON ENGEN THAYER HALL WEST POINT NY 10996-1786
1	DIRECTOR US ARMY RESEARCH LAB ATTN AMSRL CS AL TP 2800 POWDER MILL RD ADELPHI MD 20783-1145
1	DIRECTOR US ARMY RESEARCH LAB ATTN AMSRL CS AL TA 2800 POWDER MILL RD ADELPHI MD 20783-1145
3	DIRECTOR US ARMY RESEARCH LAB ATTN AMSRL CI LL 2800 POWDER MILL RD ADELPHI MD 20783-1145
	ABERDEEN PROVING GROUND
2	DIR USARL ATTN AMSRL CI LP (305)

NO. OF
COPIES ORGANIZATION

1 DIRECTOR
US ARMY RESEARCH LAB
ATTN AMSRL IS TA J GANTT
GEORGIA INST OF TCHNLGY
115 OKEEFE BLDG
ATLANTA GA 30332-0800

ABERDEEN PROVING GROUND

1 DIR USACBDCOM
ATTN: AMSCB CI JIM WOOD

25 DIR USARL
ATTN: AMSRL-CI,
W MERMAGEN
W STUREK
AMSRL-CI-H, C NIETUBICZ
AMSRL-CI-HA,
D PRESSEL (5 CP)
C ZOLTANI
AMSRL-IS-TP,
A BRODEEN
B BROOME
S CHAMBERLAIN
B COOPER
A DOWNS
G HARTWIG
R KASTE
T HANRATTY
J DUMER
A CELMINS
R HELFMAN
T KORJACK
F BRUNDICK
H CATON
M LOPEZ
L WRENCHER

USER EVALUATION SHEET/CHANGE OF ADDRESS

This Laboratory undertakes a continuing effort to improve the quality of the reports it publishes. Your comments/answers to the items/questions below will aid us in our efforts.

1. ARL Report Number/Author ARL-SR-48 (Pressel) Date of Report December 1996

2. Date Report Received _____

3. Does this report satisfy a need? (Comment on purpose, related project, or other area of interest for which the report will be used.) _____

4. Specifically, how is the report being used? (Information source, design data, procedure, source of ideas, etc.) _____

5. Has the information in this report led to any quantitative savings as far as man-hours or dollars saved, operating costs avoided, or efficiencies achieved, etc? If so, please elaborate. _____

6. General Comments. What do you think should be changed to improve future reports? (Indicate changes to organization, technical content, format, etc.) _____

CURRENT ADDRESS

_____
Organization

_____
Name E-mail Name

_____
Street or P.O. Box No.

_____
City, State, Zip Code

7. If indicating a Change of Address or Address Correction, please provide the Current or Correct address above and the Old or Incorrect address below.

OLD ADDRESS

_____
Organization

_____
Name

_____
Street or P.O. Box No.

_____
City, State, Zip Code

(Remove this sheet, fold as indicated, tape closed, and mail.)
(DO NOT STAPLE)

DEPARTMENT OF THE ARMY

OFFICIAL BUSINESS

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY MAIL
FIRST CLASS PERMIT NO 0001,APG,MD

POSTAGE WILL BE PAID BY ADDRESSEE

DIRECTOR
US ARMY RESEARCH LABORATORY
ATTN AMSRL CI HA
ABERDEEN PROVING GROUND MD 21005-5067